

11

트리 표현 ADT

목적

이번 실습에서 여러분은

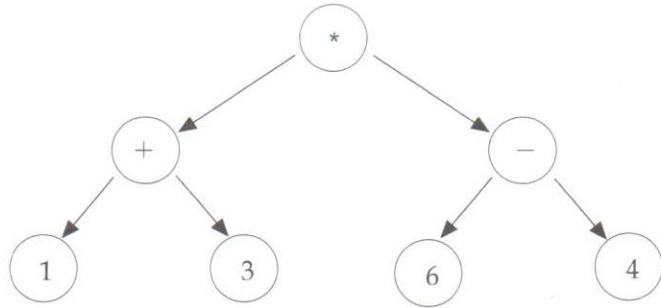
- 링크드 트리 구조를 이용하여 트리 ADT 표현의 구현을 작성한다.
- 논리적인 트리 ADT 표현의 구현을 개발하고 간단한 논리 회로 모형을 만드는 구현에 사용한다.
- Tree ADT 표현의 구현에 사용되는 전위(Preorder), 중위(inorder), 후위(postorder) 트리 운행이 어떻게 사용되는지 분석한다.

개요

여러분이 보통 수식을 선형 형태로 쓰지만 이 수식을 평가할 때 여러분들은 수식들을 계층적 객체로 다루어야 한다. 예를 들어 다음의 수식을 평가할 때

$$(1+3) * (6-4)$$

여러분은 먼저 1과 3을 더하고 6에서 4를 뺀다. 마지막으로 식의 값을 구하기 위해 두 결과를 서로 곱한다. 이들 계산 수행에서 여러분은 덧셈과 뺄셈 연산자로 이루어진 기본 위에 만들어지는 함축적 계층적 구조를 만든다. 여러분은 이 계층구조를 다음의 이진 트리를 사용하여 명백하게 표현할 수 있다. 이러한 트리는 트리 표현(expression tress)으로 언급된다.



트리 ADT의 표현

원소

트리 표현에서 각 노드는 수식 연산자나 수(numeric value)

구조

노드는 수식 연산자를 갖는 각 노드는 자식(children)을 갖는 트리를 구성한다. 각 자식(child)은 연산자의 피연산자를 표현하는 루트 노드의 서브트리이다. 수 값(numeric value)를 갖는 노드는 자식노드가 없다.

기능

ExprTree ()

요구사항:

없음

결과:

객체 생성자. 빈 트리 표현(expression tree)을 생성한다.

~ExprTree ()

요구사항:

없음.

결과:

객체 파괴자. 트리 표현을 적재하는데 쓰일 메모리 할당을 해제한다.

void build ()

요구사항:

없음.

결과:

수식을 키보드에서 전위 표기법(prefix form)으로 읽어 그에 대응하는 트리 표현을 만든다.

void expression () const

요구사항:

없음.

결과:

중위 표기법(infix form)에서 수식에 대응하는 수식을 출력.

float evaluate () const

요구사항:

트리 표현은 비어 있지 않다.

결과:

수식에 대응하는 값을 반환한다.

void clear ()

요구사항:

없음.

결과:

트리표현에서 모든 원소를 삭제한다.

void showStructure () const

요구사항:

없음.

결과:

왼쪽(root)에서 오른쪽(leaves)으로 향하는 가지를 갖는 트리 표현을 출력한다. 즉

전통적인 방향에서 시계 반대 방향으로 90도 회전한 트리가 출력된다.

트리가 비어 있으면 "Empty tree"를 출력한다. 이 연산자는 시험/디버깅을 목적으로 사용된다. 수식은 양의 정수 한 자리 수만 갖고 수식 연산자는 덧셈, 뺄셈, 곱셈, 나눗셈을 갖는다고 가정한다.

우리는 보통 수식을 중위법(**infix form**)으로 쓴다. 즉 다음의 수식처럼 각 연산자가 피연산자 사이에 위치한다.

$$(1+3) * (6-4)$$

이번 실습에서 여러분은 수식의 전위법(**prefix form**)에서 트리표현을 구성한다. 전위법에서 각 연산자는 그 연산자의 피연산자 바로 앞에 위치한다. 이전의 수식은 다음과 같이 전위법으로 쓰여진다.

$$* + 1 3 - 6 4$$

왼쪽에서 오른쪽으로 전위 표기의 수식을 수행할 때 여러분은 한정적으로 피연산자를 따르는 각 연산자를 만난다. 여러분이 연산자가 갖는 이전의 피연산자 숫자를 안다면 여러분은 트리 표현에 대응하는 산술식을 구성하는데 다음의 재귀 프로세스를 사용할 수 있다.

다음의 수 연산자나 숫자를 읽는다

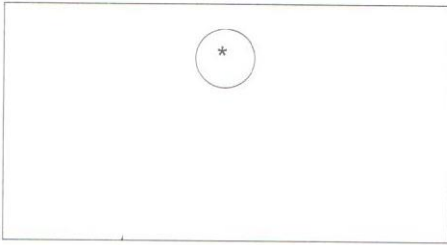
```
연산자나 숫자를 갖는 노드를 생성한다.
if 노드가 연산자를 갖으면
    then 연산자의 피연산자에 대응할 때까지
        서브트리를 재귀적으로 만든다.
    Else 노드는 잎(leaf) 노드이다.
```

여러분이 이 프로세스를 산술식에

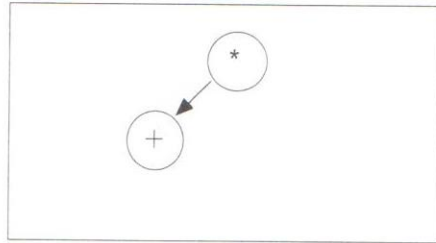
$$* + 1 3 - 6 4$$

적용한다면 트리 표현 구성은 아래와 같이 진행된다.

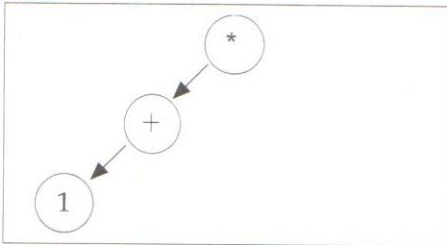
Read '*'



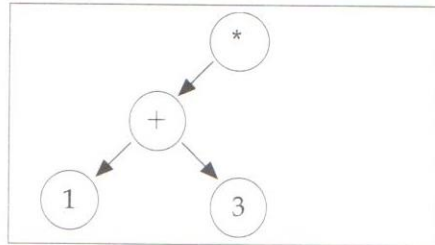
Read '+'



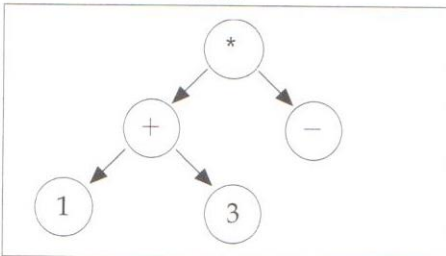
Read '1'



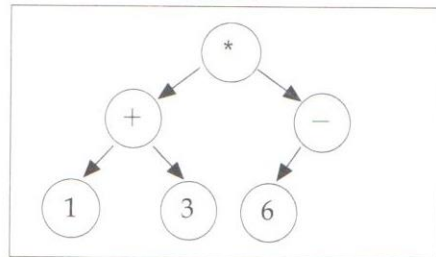
Read '3'



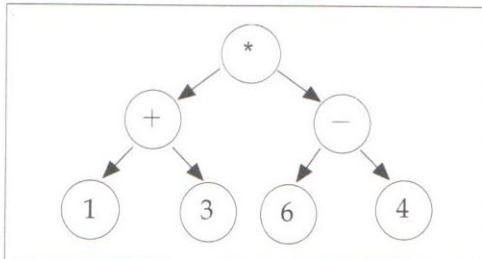
Read '-'



Read '6'



Read '4'



이 수식의 진행과정에서 모든 수 값들은 한자리이고, 음수가 아니라고 가정하고, 모든 수값은 단일 문자로 표현된다. 이 프로세스가 여러 자리수의 수를 포함하게 일반화한다면 숫자를 분리하기 위해 이 수식에서 경계자를 포함해야 할 것이다.

실습 11 : 표지

날짜_____ 구분_____

성명_____

강사가 여러분에게 할당한 다음에 연습문제들에 할당된 열에 체크표시를 하고 이 겹장을 앞으로의 연구에서 여러분이 제출할 과제물 앞에 붙이시요

	할 일	완 료
실습 전 연습	✓	
연 결 연습	✓	
실습 중 연습 1		
실습 중 연습 2		
실습 중 연습 3		
실습 후 연습 1		
실습 후 연습 2		
	총 점	

실습 11 : 실습 전 연습



날짜_____ 구분_____

이름_____

서론에서 여러분은 트리표현의 구성이 재귀를 이용하여 설명되는 것을 보았다. 이번 연습에서 여러분은 트리표현 ADT(Expression Tree ADT)에서 연산을 수행하는데 재귀함수를 이용한다.

1단계 : 트리표현 ADT에서 링크드 트리 구조를 사용하여 연산자들을 구현하라. 수식은 한자리 숫자와 음수가 아닌 수('0' - '9')와 4개의 기본적인 연산자('+', '-', '*', '/')로 구성된다고 가정한다. 각 수식은 키보드에서 모든 문자는 한 행으로 전위법(prefix form)으로 입력된다.

여러분이 실습 전 연습에서 개발한 링크드 선형 구조에서 링크드 트리구조 구현은 트리내의 노드와 트리구조 전체에 대한 두 개의 클래스를 사용한다. 트리 내의 각 노드는 문자를 포함하고(원소) 노드의 자녀(왼쪽과 오른쪽)를 가리키는 두 개의 포인터를 갖는다. 여러분의 구현은 역시 트리의 루트 노드의 포인터를 유지해야 한

 다. 파일 `exprtree.hs`에 있는 선언을 기본으로 구현하라. ShowStructure 연산의 수행  은 파일 `show11.cpp`에 있다.

```
class ExprTreeNode    // Facilitator class for the ExprTree
                      // class
{
private:

    // Constructor
    ExprTreeNode (char elem,
                  ExprTreeNode *leftPtr, ExprTreeNode *rightPtr);

    // Data members
    char element;           // Expression tree element
    ExprTreeNode *left,    // Pointer to the left child
                  *right;  // Pointer to the right child

friend class ExprTree;
```

```

};

//-----

class ExprTree
{
public:

    // Constructor
    ExprTree ( );

    // Destructor
    ~ExprTree ( );

    // Expression tree manipulation operations
    void build ( );           // Build tree from prefix
                                // expression
    void expression( ) const; // Output expression in
                                // infix form
    float evaluate( ) const; // Evaluate expression
    void clear ( );          // Clear tree

    // Output the tree structure -- used in
    // testing/debugging
    void showStructure ( ) const;

    // In-lab operations
    ExprTree (const ExprTree &valueTree); // Copy
                                            // constructor

    void commute ( ); // Commute all subexpr.


private:

    // Recursive partners of the public member functions
    // -- insert prototypes of these functions here.
    void showSub (ExprTreeNode *p, int level) const;

    // Data member
    ExprTreeNode *root; // Pointer to the root node

```


};


 **2단계** : 파일 `expmtree.hs`에 있는 `ExprTree` 클래스의 선언은 여러분의 트리표현 ADT 구현에 필요한 재귀 `private` 멤버 함수들에 대한 프로토타입을 포함하지 않는다. 이 프로토타입들을 추가하고 클래스 선언을 파일 `expmtree.h`에 저장하라.

3단계 : 여러분의 트리표현 Tree ADT 구현을 파일 `expmtree.cpp`에 저장하고 코드를 문서로 확인하라.

실습 11 : 연결 연습

날짜 _____ 구분 _____
성명 _____

여러분의 강사와 함께 여러분이 이 연습을 실습기간 동안이나 실습 중에 마칠 수 있는지 체크하라.

 파일 `test11.cpp`에 있는 시험 프로그램을 사용하여 트리표현 ADT(Expression Tree ADT)를 구현하라.

1단계 : 파일 `expmtree.cpp`에 있는 여러분의 트리표현 ADT 구현을 컴파일하라.

2단계 : 파일 `test11.cpp.dp`에 있는 시험 프로그램을 컴파일하라.

3단계 : 1, 2단계에서 생성된 오브젝트 파일을 링크하라.

4단계 : 각 수식에 대한 예상 결과를 기입하여 다음의 시험 계획을 완성하라. 여러분은 수식을 시험 계획에 추가하게 될 것이다.

5단계 : 이 시험 계획을 실행하라. 트리표현 ADT 구현에서 잘못이 발견되면 수정하고 다시 실행하라.

트리 표현 ADT에서 기능들에 대한 시험 계획			
시험 계획	수식	예상 결과	검사
한 개의 연산자	+34		
nested 연산자들	*+34/52		
처음에 모든 연산자들	-/*9321		
Uneven nesting	*4+6-75		
0을 나누기	/02		
한자리 숫자	7		